# 分组是 LINQ 最强大的功能之一。下面的示例演示如何以各种方式对数据进行分组：

## 按照单个属性。

## 按照字符串属性的首字母。

## 按照计算出的数值范围。

## 按照布尔谓词或其他表达式。

## 按照复合键。

此外，最后两个查询将它们的结果投影到一个新的匿名类型中，该类型仅包含学生的名字和姓氏。有关更多信息，请参见 [group 子句（C# 参考）](https://msdn.microsoft.com/zh-cn/library/bb384063.aspx)。

本主题中的所有示例都使用下列帮助器类和数据源。

public class StudentClass

{

#region data

protected enum GradeLevel { FirstYear = 1, SecondYear, ThirdYear, FourthYear };

protected class Student

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int ID { get; set; }

public GradeLevel Year;

public List<int> ExamScores;

}

protected static List<Student> students = new List<Student>

{

new Student {FirstName = "Terry", LastName = "Adams", ID = 120,

Year = GradeLevel.SecondYear,

ExamScores = new List<int>{ 99, 82, 81, 79}},

new Student {FirstName = "Fadi", LastName = "Fakhouri", ID = 116,

Year = GradeLevel.ThirdYear,

ExamScores = new List<int>{ 99, 86, 90, 94}},

new Student {FirstName = "Hanying", LastName = "Feng", ID = 117,

Year = GradeLevel.FirstYear,

ExamScores = new List<int>{ 93, 92, 80, 87}},

new Student {FirstName = "Cesar", LastName = "Garcia", ID = 114,

Year = GradeLevel.FourthYear,

ExamScores = new List<int>{ 97, 89, 85, 82}},

new Student {FirstName = "Debra", LastName = "Garcia", ID = 115,

Year = GradeLevel.ThirdYear,

ExamScores = new List<int>{ 35, 72, 91, 70}},

new Student {FirstName = "Hugo", LastName = "Garcia", ID = 118,

Year = GradeLevel.SecondYear,

ExamScores = new List<int>{ 92, 90, 83, 78}},

new Student {FirstName = "Sven", LastName = "Mortensen", ID = 113,

Year = GradeLevel.FirstYear,

ExamScores = new List<int>{ 88, 94, 65, 91}},

new Student {FirstName = "Claire", LastName = "O'Donnell", ID = 112,

Year = GradeLevel.FourthYear,

ExamScores = new List<int>{ 75, 84, 91, 39}},

new Student {FirstName = "Svetlana", LastName = "Omelchenko", ID = 111,

Year = GradeLevel.SecondYear,

ExamScores = new List<int>{ 97, 92, 81, 60}},

new Student {FirstName = "Lance", LastName = "Tucker", ID = 119,

Year = GradeLevel.ThirdYear,

ExamScores = new List<int>{ 68, 79, 88, 92}},

new Student {FirstName = "Michael", LastName = "Tucker", ID = 122,

Year = GradeLevel.FirstYear,

ExamScores = new List<int>{ 94, 92, 91, 91}},

new Student {FirstName = "Eugene", LastName = "Zabokritski", ID = 121,

Year = GradeLevel.FourthYear,

ExamScores = new List<int>{ 96, 85, 91, 60}}

};

#endregion

//Helper method, used in GroupByRange.

protected static int GetPercentile(Student s)

{

double avg = s.ExamScores.Average();

return avg > 0 ? (int)avg / 10 : 0;

}

public void QueryHighScores(int exam, int score)

{

var highScores = from student in students

where student.ExamScores[exam] > score

select new {Name = student.FirstName, Score = student.ExamScores[exam]};

foreach (var item in highScores)

{

Console.WriteLine("{0,-15}{1}", item.Name, item.Score);

}

}

}

public class Program

{

public static void Main()

{

StudentClass sc = new StudentClass();

sc.QueryHighScores(1, 90);

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit");

Console.ReadKey();

}

}

下面的示例演示如何通过使用元素的单个属性作为组键对源元素进行分组。在这种情况下，键是 **string**，即学生的姓氏。还可以使用子字符串作为键。分组操作将对该类型使用默认的相等比较器。

# 进行分组查询

## 1.单层分组查询

将下面的方法粘贴到 StudentClass 类中。将 Main 方法中的调用语句更改为 sc.GroupBySingleProperty()。

public void GroupBySingleProperty()

{

Console.WriteLine("Group by a single property in an object:");

// Variable queryLastNames is an IEnumerable<IGrouping<string,

// DataClass.Student>>.

var queryLastNames =

from student in students

group student by student.LastName into newGroup

orderby newGroup.Key

select newGroup;

foreach (var nameGroup in queryLastNames)

{

Console.WriteLine("Key: {0}", nameGroup.Key);

foreach (var student in nameGroup)

{

Console.WriteLine("\t{0}, {1}", student.LastName, student.FirstName);

}

}

}

/\* Output:

Group by a single property in an object:

Key: Adams

Adams, Terry

Key: Fakhouri

Fakhouri, Fadi

Key: Feng

Feng, Hanying

Key: Garcia

Garcia, Cesar

Garcia, Debra

Garcia, Hugo

Key: Mortensen

Mortensen, Sven

Key: O'Donnell

O'Donnell, Claire

Key: Omelchenko

Omelchenko, Svetlana

Key: Tucker

Tucker, Lance

Tucker, Michael

Key: Zabokritski

Zabokritski, Eugene

\*/

下面的示例演示如何通过使用除对象属性以外的某个项作为组键对源元素进行分组。在此示例中，键是学生姓氏的第一个字母。

将下面的方法粘贴到 StudentClass 类中。将 Main 方法中的调用语句更改为 sc.GroupBySubstring()。

public void GroupBySubstring()

{

Console.WriteLine("\r\nGroup by something other than a property of the object:");

var queryFirstLetters =

from student in students

group student by student.LastName[0];

foreach (var studentGroup in queryFirstLetters)

{

Console.WriteLine("Key: {0}", studentGroup.Key);

// Nested foreach is required to access group items.

foreach (var student in studentGroup)

{

Console.WriteLine("\t{0}, {1}", student.LastName, student.FirstName);

}

}

}

/\* Output:

Group by something other than a property of the object:

Key: A

Adams, Terry

Key: F

Fakhouri, Fadi

Feng, Hanying

Key: G

Garcia, Cesar

Garcia, Debra

Garcia, Hugo

Key: M

Mortensen, Sven

Key: O

O'Donnell, Claire

Omelchenko, Svetlana

Key: T

Tucker, Lance

Tucker, Michael

Key: Z

Zabokritski, Eugene

\*/

下面的示例演示如何通过使用某个数值范围作为组键对源元素进行分组。然后，查询将结果投影到一个匿名类型中，该类型仅包含学生的名字和姓氏以及该学生所属的百分点范围。使用匿名类型的原因是没有必要使用完整的 Student 对象来显示结果。 GetPercentile 是一个 helper 函数，它根据学生的平均分数计算百分比。该方法返回 0 到 10 之间的整数。

//Helper method, used in GroupByRange.

protected static int GetPercentile(Student s)

{

double avg = s.ExamScores.Average();

return avg > 0 ? (int)avg / 10 : 0;

}

将下面的方法粘贴到 StudentClass 类中。将 Main 方法中的调用语句更改为 sc.GroupByRange()。

public void GroupByRange()

{

Console.WriteLine("\r\nGroup by numeric range and project into a new anonymous type:");

var queryNumericRange =

from student in students

let percentile = GetPercentile(student)

group new { student.FirstName, student.LastName } by percentile into percentGroup

orderby percentGroup.Key

select percentGroup;

// Nested foreach required to iterate over groups and group items.

foreach (var studentGroup in queryNumericRange)

{

Console.WriteLine("Key: {0}", (studentGroup.Key \* 10));

foreach (var item in studentGroup)

{

Console.WriteLine("\t{0}, {1}", item.LastName, item.FirstName);

}

}

}

/\* Output:

Group by numeric range and project into a new anonymous type:

Key: 60

Garcia, Debra

Key: 70

O'Donnell, Claire

Key: 80

Adams, Terry

Feng, Hanying

Garcia, Cesar

Garcia, Hugo

Mortensen, Sven

Omelchenko, Svetlana

Tucker, Lance

Zabokritski, Eugene

Key: 90

Fakhouri, Fadi

Tucker, Michael

\*/

下面的示例演示如何通过使用布尔比较表达式对源元素进行分组。在此示例中，布尔表达式会测试学生的平均考试分数是否超过 75。与上述示例一样，结果被投影到一个匿名类型中，因为不需要完整的源元素。请注意，在执行查询时，该匿名类型中的属性将变成 Key 成员上的属性，并且可以通过名称进行访问。

将下面的方法粘贴到 StudentClass 类中。将 Main 方法中的调用语句更改为 sc.GroupByBoolean()。

public void GroupByBoolean()

{

Console.WriteLine("\r\nGroup by a Boolean into two groups with string keys");

Console.WriteLine("\"True\" and \"False\" and project into a new anonymous type:");

var queryGroupByAverages = from student in students

group new { student.FirstName, student.LastName }

by student.ExamScores.Average() > 75 into studentGroup

select studentGroup;

foreach (var studentGroup in queryGroupByAverages)

{

Console.WriteLine("Key: {0}", studentGroup.Key);

foreach (var student in studentGroup)

Console.WriteLine("\t{0} {1}", student.FirstName, student.LastName);

}

}

/\* Output:

Group by a Boolean into two groups with string keys

"True" and "False" and project into a new anonymous type:

Key: True

Terry Adams

Fadi Fakhouri

Hanying Feng

Cesar Garcia

Hugo Garcia

Sven Mortensen

Svetlana Omelchenko

Lance Tucker

Michael Tucker

Eugene Zabokritski

Key: False

Debra Garcia

Claire O'Donnell

\*/

下面的示例演示如何使用匿名类型来封装包含多个值的键。在此示例中，第一个键值是学生姓氏的第一个字母。第二个键值是一个布尔值，它指定该学生在第一次考试中的得分是否超过了 85。可以按照该键中的任何属性对多组值进行排序。

将下面的方法粘贴到 StudentClass 类中。将 Main 方法中的调用语句更改为 sc.GroupByCompositeKey()。

public void GroupByCompositeKey()

{

var queryHighScoreGroups =

from student in students

group student by new { FirstLetter = student.LastName[0],

Score = student.ExamScores[0] > 85 } into studentGroup

orderby studentGroup.Key.FirstLetter

select studentGroup;

Console.WriteLine("\r\nGroup and order by a compound key:");

foreach (var scoreGroup in queryHighScoreGroups)

{

string s = scoreGroup.Key.Score == true ? "more than" : "less than";

Console.WriteLine("Name starts with {0} who scored {1} 85", scoreGroup.Key.FirstLetter, s);

foreach (var item in scoreGroup)

{

Console.WriteLine("\t{0} {1}", item.FirstName, item.LastName);

}

}

}

/\* Output:

Group and order by a compound key:

Name starts with A who scored more than 85

Terry Adams

Name starts with F who scored more than 85

Fadi Fakhouri

Hanying Feng

Name starts with G who scored more than 85

Cesar Garcia

Hugo Garcia

Name starts with G who scored less than 85

Debra Garcia

Name starts with M who scored more than 85

Sven Mortensen

Name starts with O who scored less than 85

Claire O'Donnell

Name starts with O who scored more than 85

Svetlana Omelchenko

Name starts with T who scored less than 85

Lance Tucker

Name starts with T who scored more than 85

Michael Tucker

Name starts with Z who scored more than 85

Eugene Zabokritski

\*/

## 2,嵌套分组查询

# 创建嵌套组（C# 编程指南）

下面的示例演示如何在 LINQ 查询表达式中创建嵌套组。对根据学生年级创建的每个组，将根据每个人的姓名进一步划分为小组。

public void QueryNestedGroups()

{

var queryNestedGroups =

from student in students

group student by student.Year into newGroup1

from newGroup2 in

(from student in newGroup1

group student by student.LastName)

group newGroup2 by newGroup1.Key;

// Three nested foreach loops are required to iterate

// over all elements of a grouped group. Hover the mouse

// cursor over the iteration variables to see their actual type.

foreach (var outerGroup in queryNestedGroups)

{

Console.WriteLine("DataClass.Student Level = {0}", outerGroup.Key);

foreach (var innerGroup in outerGroup)

{

Console.WriteLine("\tNames that begin with: {0}", innerGroup.Key);

foreach (var innerGroupElement in innerGroup)

{

Console.WriteLine("\t\t{0} {1}", innerGroupElement.LastName, innerGroupElement.FirstName);

}

}

}

}

/\*

Output:

DataClass.Student Level = SecondYear

Names that begin with: Adams

Adams Terry

Names that begin with: Garcia

Garcia Hugo

Names that begin with: Omelchenko

Omelchenko Svetlana

DataClass.Student Level = ThirdYear

Names that begin with: Fakhouri

Fakhouri Fadi

Names that begin with: Garcia

Garcia Debra

Names that begin with: Tucker

Tucker Lance

DataClass.Student Level = FirstYear

Names that begin with: Feng

Feng Hanying

Names that begin with: Mortensen

Mortensen Sven

Names that begin with: Tucker

Tucker Michael

DataClass.Student Level = FourthYear

Names that begin with: Garcia

Garcia Cesar

Names that begin with: O'Donnell

O'Donnell Claire

Names that begin with: Zabokritski

Zabokritski Eugene

\*/

# 对分组操作执行子查询（C# 编程指南）

本主题演示执行以下任务的两种不同方式：创建一个查询，以便将源数据排序到不同的组中，然后分别对每个组执行子查询。每个示例中的基本技术都是使用一个名为 newGroup 的“延续”对源元素进行分组，然后生成一个针对 newGroup 的新的子查询。此子查询是针对外部查询所创建的每个新组运行的。请注意，在此特定示例中，最终的输出不是组，而是匿名类型的平面序列。

有关如何分组的更多信息，请参见[group 子句（C# 参考）](https://msdn.microsoft.com/zh-cn/library/bb384063.aspx)。

有关延续的更多信息，请参见 [into（C# 参考）](https://msdn.microsoft.com/zh-cn/library/bb311045.aspx)。下面的示例使用内存中数据结构作为数据源，但相同的原理适用于任何种类的 LINQ 数据源。

public void QueryMax()

{

var queryGroupMax =

from student in students

group student by student.Year into studentGroup

select new

{

Level = studentGroup.Key,

HighestScore =

(from student2 in studentGroup

select student2.ExamScores.Average()).Max()

};

int count = queryGroupMax.Count();

Console.WriteLine("Number of groups = {0}", count);

foreach (var item in queryGroupMax)

{

Console.WriteLine(" {0} Highest Score={1}", item.Level, item.HighestScore);

}

}